CS 184: Assignment 1 — Transformations

Ravi Ramamoorthi

Goals and Motivation

The purpose of the homework is to *fill in the code to allow rotation of the viewpoint around a scene.* This homework is to be done individually.

The assignment was designed with 2 main goals, as a gentle introduction to the course and initial material on transformations. The first goal is to understand how viewing and other transformations are used to render scenes. You will implement a simple crystall ball interface, standardly used for viewing, and also see how to build up the standard transformation matrices for this purpose. The second goal is to get you to compile and run an openGL program so you can make sure you have everything set up correctly for later assignments (all assignments, except most of the last raytracing assignment will require the use of OpenGL).

This assignment also gives you a chance to play around a little and explore. In particular, the assignment skeleton is written in OpenGL. While not absolutely required for this assignment, it would help for later assignments (which will require more extensive use of OpenGL) if you could at least understand the framework—we'll discuss this briefly in class—and you may also want to play around with it a little. However, do note for the final submission, the requirement is to *not submit any modifications to the main assignment framework.* This requirement is to provide standardization and for grading.

While the actual coding work is minimal, some thinking is needed, so you will want to start early. Also note that one goal is to have your system set up so you can compile and run OpenGL programs for later. Debugging this issue early on is crucial for smooth operation of the rest of the course.

Post any questions you have to the newsgroup, since other students will want to see the answers too. However, do not post anything resembling code.

Getting Started

In principle, OpenGL and related code should be portable across many platforms, and indeed this is its strength. As such, you may use any computer and platform you choose, but will need some kind of C++ development framework and an installation of OpenGL and Glut (more on the latter below). Please speak with the TA if you need a C/C++ development environment such as Microsoft's Visual Studio.

In practice, this assignment was created primarily for and with Windows software using C++ Visual Studio. The code should be fairly portable and I have tested it on my Linux system (my compile command was *g++ -g *.cpp -lGL -lglut -o hw1* and I needed to change the include line in *main.cpp* to capital *GL/gl.h* instead of *gl/gl.h*). However, if you plan to use a Mac or UNIX, start early so you have time to contact the system administrator about problems you encounter (such as setting up glut, or finding the directory where gl.h is located). Also note that the solution provided is a Windows executable (but see below about simply using 'g' to toggle).

You probably already have openGL set up on your computer. Search for gl.h to see if you have it. If not, go to *www.opengl.org* and follow the instructions on downloading the appropriate files. You probably don't already have Glut set up. If you want, you can also find the download for that package from *www.opengl.org* (just google for GLUT to find many sources for it). We'll also provide a link on the assignment website to download the three necessary glut files. You will have to set up your path to point to the location of the dll. Note that some people have had problems compiling the skeleton with the new glut version. The one in the zip file we provide should work.

If you want more details of OpenGL, there are many websites, some of which are linked to from the main course page, and others that can be obtained by googling. The website Nehe, at *http://nehe.gamedev.net/lesson.asp?index=01* provides useful OpenGL lessons, especially involving setting up OpenGL in various environments.

Skeleton Code

Next, download the assignment (we will provide a link to the zip file on the class website). Unzip *hw1.zip*. Run *hw1_solution_v2.exe*. If you get an error message about the glut dll, try adjusting your path, or simply copying it into the same directory.

Now, compile and run the files as is.

You should see the same teapot as in the solution on a blue background, but the arrow keys won't work. Your job will be to make them work.

For the final submission **only make changes to** *Transform.cpp*. Do not add any *#include* lines to the code. Do not change *Transform.h*. If you do make changes to any of the other files, make sure your solution works without those changes. You will only be submitting *Transform.cpp*. The files *nv_mathdecl.h*, *nv_algebra.cpp*, *nv_math.h* and *nv_algebra.h* are provided as helper files. Don't worry about the details of these files. Concentrate on *main.cpp*. That said, you will probably have a much easier time if you look at the function *set_rot()*.

Once again, if you have difficulty compiling and setting up the skeleton code, please contact the TAs as soon as possible. The skeleton code was originally written quite some time back, and we cannot easily predict incompatibilities with the diverse sets of machines in current use. It is likely that any issues relevant to you will be to others as well. We want to get over the initial hump of compiling a program as soon as possible, so we can focus on the course material.

Assignment Specification

You will be implementing a classic crystal ball interface. This simulates a world in which the viewer is glued to the outside of a transparent sphere, looking in. The sphere is centered at the origin, and that is the direction towards which your eye is always pointing. At the origin, there is something interesting to look at, in this case, a teapot.

You can change the viewpoint by rotating the crystal ball in any direction about the origin. Usually this is done with a mouse, but you will be using the keyboard for this assignment to make things easier. You must think about how the position of the eye and direction of the up vector change with left-right or up-down rotations.

Fill in the parts of *Transform.cpp* that say "//FILL IN YOUR CODE HERE". First, you should fill in *left()* and *up()*. Once these are working, fill in *lookAt()*. The compiled correct solution has been provided for you. Your solution must behave identically to *hw1_solution_v2.exe*. All you need to test this is to press the arrow keys a few times and toggle 'g' to make sure your *lookAt()* works. (The latter verification should work even for those on platforms that cannot run the pre-compiled solution).

Separately from the main assignment, you should also play around with the solution to try and make the red and blue light line up by updating one and not the other (press 'h' when the solution is running). Also, you will have to answer a question in the submission email (see below).

Submission Instructions

We will be using the submit software for submission of this assignment. Instructions for using the submission software are at *http://inst.eecs.berkeley.edu/cgi-bin/pub.cgi?file=submit.help*. Run "submit as1" from a directory containing the following:

- *Tranform.cpp*
- *Answer.txt* that answers the following question. Describe why it is that the red light seems to provide more than just a point light source, illuminating the entire half of the teapot, while the blue light leaves most of the teapot dark, only illuminating as a point source. This requires some information about OpenGL. The answer is found in the red book (Woo), on page 189 (4th ed, 5th ed, 6th ed = 195, 7th ed = 214-215).

If you did not use Visual Studio on a Windows platform, please describe what you did to complete this assignment (can put this in *Answer.txt*), and any special tweaks so we can make them available for future iterations of the course. However, regardless of your own development environment, please do try to ensure your assignment can be compiled and run on a Windows environment with Visual C++. This is likely what the TAs will grade on.

Documentation and Hints

We include below some optional documentation and hints that may be of interest. You are not required to use or refer to any of the material here, however. This mainly pertains to the material needed in *Tranform.cpp*. *Main.cpp* is not very well documented; one goal is for you to try to learn some OpenGL if you want to look at it in more detail. You may want to augment it, for example adding a quit command. However, your final submission should only include *Transform.cpp*.

Helper Functions: In the course of modifying *Transform.cpp*, you may want to make use of the helper classes and functions from the nv libraries. These include *vec3* for 3-vectors, *mat3* and *mat4* for 3×3 and 4×4 matrices, normalize to set vectors to unit norm, and *set_rot* to specify rotations. You can consult the sources on these functions. If you do choose to use these functions, one common gotcha is that matrices are assumed column major in the nv libraries as opposed to the standard math convention of row major.

Left: The simplest function to fill in is left. The input is the degrees of rotation, the current eye 3-vector and current up 3-vector. Note that you may need to convert degrees to radians (in the standard way) to set up a rotation matrix. Your job is to update the eye and up vectors appropriately for the user moving left (and equivalently right). This function should not require more than about 3 lines of code to do the appropriate rotations.

Up: The up function is slightly more complicated, but satisfies the same basic requirements as left. You might want to make use of helper functions like *cross* (get the syntax correct) and auxiliary vectors. Again, you need to update the eye and up vectors correctly.

lookAt: Finally, you need to code in the transformation matrix, given the eye and up vectors. You will likely need to refer to the class notes to do this. It is likely to help to define an xyz coordinate frame (as 3 vectors), and to build up an auxiliary 4×4 matrix M which is returned as the result of this function. Consult class notes and lectures for this part.

Acknowledgements

This assignment was developed with Aner Ben-Artzi (a former Berkeley undergrad) a few years ago. A few clarifications have since been added.